REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 22, 1997	Interim 1/1/96 thru 2/28/97

4. TITLE AND SUBTITLE
"Applications and Systems for Large-Scale Adaptive Parallelism"

5. FUNDING NUMBERS
N00014-96-1-0328
PR# 96PRO2521-00

6. AUTHOR(S)
David Gelernter

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Suzanne K. Polmar, Director
Grant and Contract Administration
Yale University
12 Prospect Place
New Haven, CT. 06511 - 3516

8. PERFORMING ORGANIZATION REPORT NUMBER
622A-31-42282

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research Regional Office Boston
495 Summer Street Room 103
Boston, MA. 02210-2109

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
N00014-96-1-0328

11. SUPPLEMENTARY NOTES
None

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Our research over the last year has focussed on building a Java-based Linda system as a basis for the wide-area adaptive parallelism (AP) system we anticipated in our original proposal. Java allows us to attack one of the biggest problems of AP in a WAN environment, heterogeneity; AP requires that an ongoing parallel computation be able to acquire (and drop) nodes dynamically, which requires in turn that the AP application be capable of running on any node in the pool. An application written in Java can be executed on any machine with a resident Java environment. The obvious main problem is performance; the performance of interpreted Java can't be expected to compete with the performance of compiled Fortran or C or C++. It seems clear, though, that Java's performance limitations will tend to disappear as dynamic and ordinary compilers become available. We anticipate that security also will be an important issue for AP (how do we protect resource donors from a rogue AP application–and AP applications from a rogue donor?); by working with Java, we can make use of an existing, decent security infrastructure, and future work that will go into improving the structure as weaknesses are identified and new threats are encountered. We have made considerable progress designing and implementing this Java-based system.

14. SUBJECT TERMS	15. NUMBER OF PAGES
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Progress Report
ONR Grant #N00014-96-1-0328

Our research over the last year has focussed on building a Java-based Linda system as a basis for the wide-area adaptive parallelism (AP) system we anticipated in our original proposal. A central problem in AP in WAN environment has always been the issue of heterogeneity; AP requires that an ongoing parallel computation be able to acquire (and drop) nodes dynamically, which requires in turn that the AP application be capable of running on any node in the pool. We had assumed at the start that we would handle heterogeneous execution by means of multiple compilations: before dropping an application into a Piranha pool (Piranha being our AP programming environment), the host node (the application's node-of-origin) would assemble a list of all potential hosts, and arrange for the creation of one version of the executable for every type of node. At runtime the Piranha system would, when aquiring a node of type T for application Q, withdraw an executable for Q of type T from the "executables library" that had been created beforehand. This strategy is still workable in principle, but the management burden it imposes is large. It requires that each potential host have access to a collection of compilers or cross-compilers, that libraries of executables be maintained properly, and so on. The Java environment represents, potentially, an attractive alternative; an application written in Java can be executed on any machine with a resident Java environment.

The obvious main problem is performance; the less-obvious secondary problem has to do with backward compatibility. The performance of interpreted Java can't be expected to compete with the performance of compiled Fortran or C or C++. It seems clear, though, that Java's performance limitations will tend to disappear as dynamic and ordinary compilers become available. The backward compatibility has to do with existing AP programs; a fair number of applications have been developed for existing Piranha systems using C- or Fortran-based Linda systems. A WAN Piranha environment that doesn't support these applications is obviously handicapped to a point, but compatibility issues shouldn't constrain research, especially with AP programming in such a preliminary state of development. We also anticipate that security is likely to be an issue for adaptive parallelism. How do we protect resource donors from a rogue Piranha application—and a Piranha application from a rogue donor? Security has proven be a hard problem in general for internet activities. By working with Java, we can leverage an existing, decent security infrastructure, as well as the work that will go into improving that structure as weaknesses are identified and new threats are encountered.

We have made considerable progress designing and implementing the Java-based system. The major issues are source-level language support and the runtime libraries that implement coordination mechanisms (the coordination mechanisms of Piranha are based on Linda mechanisms). Java itself suggests an obvious attack on source-level support in terms of the class mechanisms; we

can define a "tuple space" (the shared virtual object memory that underlies coordination in Linda) in terms of a class whose methods implement the fundamental tuple space communication operations. We can define a "tuple," an object (in other words) for insertion in or removal from a tuple space, in terms of a class also. To create a tuple space, we instantiate a tuple space class and bind it to a tuple-space-type name; to insert a tuple, we instantiate the appropriate tuple class and use the "insert" method ("out" in Linda) defined by the tuple space, with the tuple object itself as the argument. We read or remove tuples in essentialy the same way; we define a template (or "anti-tuple") as a class, and instantiate the class to a template as needed, with the null value bound initially to any formals. Values from the tuple are bound to such "formal" fields after the tuple-read or -removal operation is complete.

Performance of the existing system is limited by interpreted Java performance, but we expect Java performance to improve. Our main research emphasis shifts now to the runtime system, where we expect to be able to re-use a signifiant proportion of existing code from our LAN AP systems. We also plan to begin investigating issues related to making more intelligent use of Java's security infrastructure, such as the class loader and the applet/application distinction.

Finally, we will begin to address the challenge of managing the highly dispersed and heterogeneous collection of resources encompassed by the typical WAN. This will involve merging our existing Trellis work with the newly implemented Java-based ap system.